Beskrivelse

Vi har utviklet en app sterkt inspirert av brettspillet Ludo. I appen kan du spille ludospill med fire spillere, der spillerne kan være mennesker eller roboter. Du kan for eksempel spille med fire menneskelige spillere og ingen roboter, eller med en menneskelig spiller og tre roboter. Brukerne kan selv velge navnet på Ludo-spillet og navnene til de menneskelige spillerne. Robotspillerne har et enkelt design, da de flytter en tilfeldig brikke hvis de har et lovlig trekk. Vi har også implementert en tenkepause på ett sekund for robotene før de gjør et trekk, slik at trekkene deres ikke skjer umiddelbart.

Ludo-appen følger de klassiske reglene, inkludert knockout av motstanderbrikker, dobbeltkast ved å rulle en sekser, maksimum tre seksere på rad, og at ludo-brikker ikke kan passere tårn. Spillere kan også lagre et pågående spill til en fil og laste det inn igjen senere. Dette gir spillere muligheten til å fullføre et Ludo-spill over flere økter, men også muligheten til å sikre at spilldataene ikke går tapt.

Spillet er designet i 2D og interaksjonen er basert på museklikk. Appen inneholder også bakgrunnsmusikk hentet fra Super Mario. I startmenyen kan spillerne velge om de ønsker å aktivere eller deaktivere musikken.

Diagram

Diagrammet blir presentert på siste side. Det er et klassediagram som inneholder alle klassene i appen utenom kontrollerne.

Spørsmål

1)

Vi har brukt et bredt spekter av pensum. Vi har brukt interface på flere måter. En måte er Collection, ved tilfeller hvor vi kun har behov for begrensede metoder til ArrayList. Interface er også brukt i kombinasjon med observasjon/observert teknikken. Vi bruker for eksempel observasjon/observert for å holde kommunikasjonen mellom GameEngine og

GameFaceController, slik at f.eks. en popup kan bli vist i gameFace når gameEngine finner ut av at en spiller har vunnet spillet.

GameEngine implementerer også et interface, og dette interfacet gjør at det blir mulig for gameEngine å vite når en popup er synlig i gameFace. Når en popup er synlig i gameFace, så venter robotene med å gjøre trekk, helt til popupen i gameFace blir lukket.

Arv er brukt i flere deler av koden. Det beste eksempelet er i utvidelse av Player, hvor RobotPlayer har det meste av logikken til Player, men noe annen logikk, samt ekstra metoder, som kun RobotPlayer har, som f.eks. makeRobotMove(), som har logikk for hvilken brikke som skal flyttes (en tilfeldig brikke, som har et lovlig trekk).

Delegering og streams brukes flittig, og dette kan vi se mange eksempler på i player-klassen. For eksempel, så delegeres ansvaret for å hente de forskjellige brikkene på brettet i getPieces() metoden til Player-objektene, ved hjelp av en stream. Vi har også brukt en hele rekke med lambdauttrykk overalt i koden, i tilknytning til streams. For eksempel så lager vi en egen komparator ved hjelp av et lambdauttrykk, for å sortere spillerne i konstruktøren til gameEngine.

Vi har også en god del validering i vårt ludospill, og dette gjør at det i praksis er umulig å finne en sekvens med inputs som gjør at spillet krasjer. Vi har også fått til feedback, i henhold til Don Normans prinsipper om godt grensesnitt-design. For eksempel, hvis du prøver å legge inn et spillernavn som har mer enn 9 bokstaver, så vil det vises tekst på skjermen som forteller deg at en spillers navn maks kan inneholde 9 bokstaver.

2)

Vi tenker egentlig at vi har brukt alt som er av pensum i emnet. Vi har ikke laget egne comparators og iteratorer, men det har vi egentlig ikke hatt bruk for. Vi har heller ingen abstrakte klasser, men det har vi heller ikke hatt bruk for.

3)

Vi har prøvd så godt vi kunne å følge Model-View-Controller-prinsippet (MVC), men vi ser at vi kunne ha fulgt prinsippet bedre.

Vi kunne for eksempel slettet relasjonen mellom GameFaceController og (SaveAndReadToFile + GameNameInfo), for da ville kontrolleren i større grad fungert som en mellommann mellom backend-filene og gameFace.fxml.

Det blir også litt mye logikk i gameFaceController, for eksempel i gameSetup() og loadGameSetup(). Vi kunne sannsynligvis ha laget en egen klasse som gjorde mesteparten av jobben her.

Vi vil konkludere med å si oss rimelig fornøyd med hvordan vi fikk til MVC-prinsippet. Noe kunne vært forbedret, men vi har også fått til en god del.

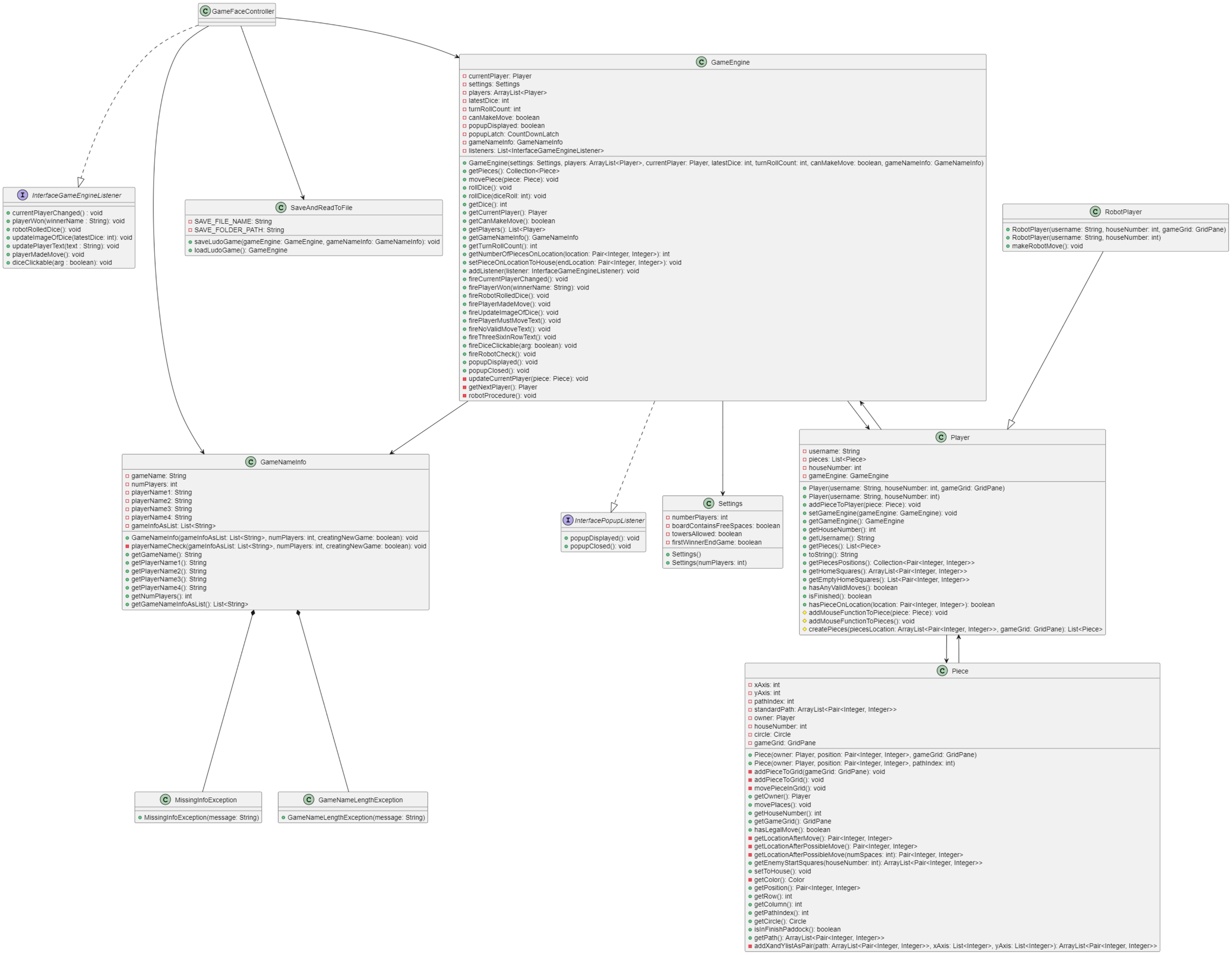
4)

Vi har i hovedsak brukt tre forskjellige teknikker for å debugge underveis og etterpå. I starten var det mest print-setninger som ble brukt, for å se at forventet resultat var det man fikk ut på skjermen. Dette var på et stadium da spillet ikke var spillbart, men man ønsket å teste deler av koden.

Etterhvert som spillet fungerte i større grad og koden ble mer omfattende, foregikk mye av debuggingen ved å prøve forskjellige funksjonaliteter og å lese eventuelle feilmeldinger. I tillegg gjorde vi debugging ved å bare spille et ludospill, og undersøke om en brikke flytter til det stedet brikken skulle flytte til, og at det ble riktig person sin tur osv. Hvis en brikke for eksempel flyttet til feil sted, så noterte vi det, og fant ut av hva som var galt i koden.

I siste fase har vi laget Junit-tester for å teste mange ulike scenarioer. Scenarioene blir ikke like avanserte som de man får av å bare spille et vanlig ludospill. Vi kan ikke teste alt i våre tester, men vi har valgt å teste om grunnleggende funksjonaliteter fungerer. Et eksempel er at vi tester at en spiller mister sitt tredje trekk dersom de kaster 3 seksere på rad. Vi har egentlig

tenkt at vi finner ut av om spillet ikke kjører riktig dersom vi spiller en runde ludo, så vi har gått for relativt simple enhetstester. Vi har prioritert tester som sjekker om et opprettet spill faktisk fungerer, og at et slikt spill kan lagres, for spillernavnvalidering vet vi fungerer ved at vi har testet appen. I siste fase fant vi faktisk noen feil, som at det ikke gikk an å lagre et spill dersom det ikke var gjort noen trekk. Disse feilene fikk vi rettet opp i.


GameFaceController
InterfaceGameEngineListener
currentPlayerChanged() : void
playerWon(winnerName : String): void
robotRolledDice(): void
updateImageOfDice(latestDice: int): void
updatePlayerText(text : String): void
playerMadeMove(): void
diceClickable(arg : boolean): void
SaveAndReadToFile
SAVE_FILE_NAME: String
SAVE_FOLDER_PATH: String
saveLudoGame(gameEngine: GameEngine, gameNameInfo: GameNameInfo): void
loadLudoGame(): GameEngine
GameEngine
currentPlayer: Player
settings: Settings
players: ArrayList<Player>
latestDice: int
turnRollCount: int
canMakeMove: boolean
popupDisplayed: boolean
popupLatch: CountDownLatch
gameNameInfo: GameNameInfo
listeners: List<InterfaceGameEngineListener>
GameEngine(settings: Settings, players: ArrayList<Player>, currentPlayer: Player, latestDice: int, turnRollCount: int, canMakeMove: boolean, gameNameInfo: GameNameInfo)
getPieces(): Collection<Piece>
movePiece(piece: Piece): void
rollDice(): void
rollDice(diceRoll: int): void
getDice(): int
getCurrentPlayer(): Player
getCanMakeMove(): boolean
getPlayers(): List<Player>
getGameNameInfo(): GameNameInfo
getTurnRollCount(): int
getNumberOfPiecesOnLocation(location: Pair<Integer, Integer>): int
setPieceOnLocationToHouse(endLocation: Pair<Integer, Integer>): void
addListener(listener: InterfaceGameEngineListener): void
fireCurrentPlayerChanged(): void
firePlayerWon(winnerName: String): void
fireRobotRolledDice(): void
firePlayerMadeMove(): void
fireUpdateImageOfDice(): void
firePlayerMustMoveText(): void
fireNoValidMoveText(): void
fireThreeSixInRowText(): void
fireDiceClickable(arg: boolean): void
fireRobotCheck(): void
popupDisplayed(): void
popupClosed(): void
updateCurrentPlayer(piece: Piece): void
getNextPlayer(): Player
robotProcedure(): void
RobotPlayer
RobotPlayer(username: String, houseNumber: int, gameGrid: GridPane)
RobotPlayer(username: String, houseNumber: int)
makeRobotMove(): void
GameNameInfo
gameName: String
numPlayers: int
playerName1: String
playerName2: String
playerName3: String
playerName4: String
gameInfoAsList: List<String>
GameNameInfo(gameInfoAsList: List<String>, numPlayers: int, creatingNewGame: boolean): void
playerNameCheck(gameInfoAsList: List<String>, numPlayers: int, creatingNewGame: boolean): void
getGameName(): String
getPlayerName1(): String
getPlayerName2(): String
getPlayerName3(): String
getPlayerName4(): String
getNumPlayers(): int
getGameNameInfoAsList(): List<String>
InterfacePopupListener
popupDisplayed(): void
popupClosed(): void
Settings
numberPlayers: int
boardContainsFreeSpaces: boolean
towersAllowed: boolean
firstWinnerEndGame: boolean
Settings()
Settings(numPlayers: int)
Player
username: String
pieces: List<Piece>
houseNumber: int
gameEngine: GameEngine
Player(username: String, houseNumber: int, gameGrid: GridPane)
Player(username: String, houseNumber: int)
addPieceToPlayer(piece: Piece): void
setGameEngine(gameEngine: GameEngine): void
getGameEngine(): GameEngine
getHouseNumber(): int
getUsername(): String
getPieces(): List<Piece>
toString(): String
getPiecesPositions(): Collection<Pair<Integer, Integer>>
getHomeSquares(): ArrayList<Pair<Integer, Integer>>
getEmptyHomeSquares(): List<Pair<Integer, Integer>>
hasAnyValidMoves(): boolean
isFinished(): boolean
hasPieceOnLocation(location: Pair<Integer, Integer>): boolean
addMouseFunctionToPiece(piece: Piece): void
addMouseFunctionToPieces(): void
createPieces(piecesLocation: ArrayList<Pair<Integer, Integer>>, gameGrid: GridPane): List<Piece>
Piece
xAxis: int
yAxis: int
pathIndex: int
standardPath: ArrayList<Pair<Integer, Integer>>
owner: Player
houseNumber: int
circle: Circle
gameGrid: GridPane
Piece(owner: Player, position: Pair<Integer, Integer>, gameGrid: GridPane)
Piece(owner: Player, position: Pair<Integer, Integer>, pathIndex: int)
addPieceToGrid(gameGrid: GridPane): void
addPieceToGrid(): void
movePieceInGrid(): void
getOwner(): Player
movePlaces(): void
getHouseNumber(): int
getGameGrid(): GridPane
hasLegalMove(): boolean
getLocationAfterMove(): Pair<Integer, Integer>
getLocationAfterPossibleMove(): Pair<Integer, Integer>
getLocationAfterPossibleMove(numSpaces: int): Pair<Integer, Integer>
getEnemyStartSquares(houseNumber: int): ArrayList<Pair<Integer, Integer>>
setToHouse(): void
getColor(): Color
getPosition(): Pair<Integer, Integer>
getRow(): int
getColumn(): int
getPathIndex(): int
getCircle(): Circle
isInFinishPaddock(): boolean
getPath(): ArrayList<Pair<Integer, Integer>>
addXandYlistAsPair(path: ArrayList<Pair<Integer, Integer>>, xAxis: List<Integer>, yAxis: List<Integer>): ArrayList<Pair<Integer, Integer>>
MissingInfoException
MissingInfoException(message: String)
GameNameLengthException
GameNameLengthException(message: String)